Test case prioritization for black box testing essay

Component based system development offer great flexibility and improvements in large software systems development.

Developer of system needs not to start from scratch, but by using Commercial-Off-The-Shelf (COTS) component he can integrate the whole system applying the glue code with component interface. Most of the COTS-components are black box in nature because of source code unavailability, which makes the testing process difficult in development leading to greater testing time and higher cost.

Using the test case prioritization technique, we can reduce the cost and time of testing. Most of the test case prioritization techniques proposed in previous works applies source code information as prioritization factors because those are based on OOT technology. In CB-system, it is proved difficult to apply those direct measurement factors for testing.

Here, in this paper, I propose test case prioritization technique for component based system.

The technique is black box in nature because it incorporates indirect measure of factors, namely; 1) Criticality of the Component (COC) and 2) Fault Proneness of Component (FPcomp). Criticality of the Component is the direct product of Component Dependency Density and Functional Importance of Component. I have proposed “ Component Dependency Density (DDcomp) matrix” for the component and it can be measured using Component Dependency Graph. After, prioritization value for each test case can be computed using the “ Component Prioritization Value (CPV) matrix”.

As a proof of concept I conducted a simulated case study with ATM system and testing process is performed using three different priority schemas.

Case study and results prove the usefulness of the technique in reducing cost of testing and improving reliability of system detecting the severe faults at early stage of testing. Key Words: Component based system, black box testing, Test Case prioritization technique, Criticality of Component, Fault proneness, Component Dependency Density ACKNOLEDGEMENT

First, I offer my sincerest gratitude to my supervisor, Associate Professor Richard Lai, who has supported me throughout my thesis with his patience and knowledge. In addition, his constant feedback, encouragement, guidance and support at every phase of writing this thesis from the initial to the final level enabled me to develop an in-depth understanding of the research area. Second, I am thankful to Associate Professor Ben Soh, who has encouraged me for this thesis, provided me the guidance in thesis writing and empowered my critical reading and writing skills.

Component based software engineering (CBSD) has become recognized as such a new discipline of software engineering, offering convenience as well as fast design/implementation. Component-based software engineering (CBSE) shifts the emphasis from programming software to composing software systems. Opposite to OO-techniques, which are white box in nature, COTS components are black box in nature due to unavailability of source code, resulting in complexity of testing, verification and validation. Without source code, many OO testing techniques will not be applicable.

The greatest challenges in component-based software development are that suitable ready-made components are difficult to find, and if they are found, they are not necessarily of good quality. In addition, components have different dependencies between them, source code is seldom available, and the target context of the components is unknown. The component developer tests the component for its correct functionality and implementation logic via unit testing. When, system integrator integrates the system combining number of components, the focus of testing is the required functionality supposed to provide by the component.

Integrator tests the functional aspect of subject component. System Integration tester uses integration testing and system testing methods to find the faults. In CB, testing, main focus is on interfaces, which are only external parts of components. Interface supposedly suffers from component synchronization problems, coupling complexity and runtime events generated by the integration of components. In previous work, a number of techniques were proposed for testing component-based systems, among which most are code based or model based.

Same time, testing is the most time and resource consuming process in any software development.

To reduce the cost of testing without decreasing the effectiveness is the critical management problem for today’s entrepreneurs. Answer is to prioritize the test cases so, which are more important run earlier in test process. There can be different goals of the test case prioritization such as increasing reliability or performance of system, reducing cost per test case.

Based on the aim of testing, the prioritization factors are selected and test cases are sort out using those factors. Actually, regression testing is the most time /cost consuming process, but important aspect of testing.

Many techniques were proposed in OO domain to prioritize the test cases for regression testing. Those techniques are code based and difficult to apply in component based development environment. In OO, measure of factors directly related to complexity and fault severity are used, where as in CBS we hardly have such measures, so we have to depend on the indirect measures such as fault proneness of component or complexity of integration. In CB-system, it is obvious that as the number of interfaces used for component integration increases the complexity of system integration increases.

One component couples with more than one component to provide specific functionality.

As the dependency of component is higher, the component is more prone to fault. Fault severity is internal characteristic of component dependency. Here, this paper proposes a test case prioritization technique in component environment. The factors I am using are criticality of component and fault proneness of component. Based on above factors the metrics/equation of criticality of component is proposed to measure the importance of component for testing purpose.

Case study is provided to evaluate effectiveness of techniques.

The paper is organized as below: section 1 and 2 provides the introduction and background related to object oriented techniques and their limitation compared to component based system development techniques. Section 3 presents survey of previous techniques for test case prioritization based on applied prioritization factors. Section 4 describes the various complexity metrics for component based system and makes comparison between them. Section 5 compares the various techniques and metrics to measure the fault proneness.

Section 6 presents the proposed technique for test case prioritization in component based black box testing. New matrix called Dependency density of Component (DDcomp) is proposed and based on that proposed technique is described which is based on two prioritization factors namely: 1) Criticality of Component and 2) Fault Proneness of Component interactions.

Section 7 presents the experimental case study for validation of proposed technique. Section 7. 1, 7. 2, and 7. 3 orderly present the system under test, experiment setting/ factor collection process and analysis of result.

At last, section 8 presents the conclusion and future work.

2. BACKGROUND Object oriented technology and software design principals entered in the mainstream software development to mirror /mimic the behaviour of the complex real word systems outside. The root principals of the OO techniques are incorporating states and behaviour of the subject system. Therefore, it heavily rely on the how objects are connected with each other rather than Why? Examples relations are associations,

inheritance, polymorphism, and uses, which most of the times, shadows the understating of why object are relates to each other.

In Response to above phenomenon , the Object technology abstract out the continuous nature of real world systems by incorporating time and space in discrete quantities. Component based technology, a subset, special case of Object oriented technology views each logical particle of the software system as a unique component. Component can be a class, object, module or subsystem, which functions independently irrespective of the other component of the system, easing the development process of large complex software. We can build the system by combining the components from different venders at reduced cost and time.

However, the views of the both techniques are different by nature: an object technique relay on relation between objects and is white box in nature.

Contrast CB systems, which relays on functionality of the component, making the component as black box. Due to unavailability of source code, the white box testing of COTS –component is nearly impossible that is why developer has to depend on functional black box testing. There are numbers of methods, techniques and, frameworks for testing both OO and CB systems, which are used at various level of software development process such as requirement capturing, design and coding.

When using in the special cases such as embedded real-time systems or component based systems , which has special problem space and development requirements the limits of OO technique come in to focus and developer has to work keeping in the mind the limitations of Object oriented techniques. The real world behave as continuum[1]. Time and space are the un-abstract-able parameters in real world systems.

The status of the system at particular time instance depends on the one or more events occur simultaneously on component /object, so it is relative in nature [2].

Opposite of above fact, the OO technique cannot capture the object’s continuous behaviour at flow of time. For Examples , there are number of observations noted in the papers[1] such as interaction of electric and magnetic field, thermal conductivity in thermodynamics , and gyroscopic effects on spinning objects , which lights the limitations of object techniques. Focusing on all above examples, component based techniques most suites the problem characterization, where the completeness of integration is most important.

CB technique provides us not fully but partial flexibility to define relative time measures at interaction of component, but it is out of focus of the paper here. Adding to this, when modelling the OO-system , OO-models, both static and behavioural, focuses on system modelling issues, but researches and surveys indicated that the development processes are equally as prevalent as modelling difficulties[3].

Again, the Object oriented technique is by nature white box, source code availability allows us to test how object interacts with one another.

Statements, functions, classes, relations, implementation logic are the most focused points when testing the OO-Systems. Opposite to this, Commercial-off-the-shelf (COTS) components are black box in the nature. System integrator has not any knowledge of logic or, implementation of component, making impossible to white-box test the component. Now, the integrator can only access the interface/port of the components to verify the validity of component has required functionality.

Method of Black box testing is most promising approach when testing the CB-Systems in such environment. . 2. COMPONENT BASED DEVELOPMENT In the component-based software development paradigm, large software is built by assembling pre-built and independently developed “ plug and play” type software parts, called software components.

The desired system behaviour is achieved through the collaborative actions of the assembled components[4]. Figure 1 : Comparison of traditional and component based system development (http://www. elainetron. com/images/12stewart01.

esp. art. gif) Component-based systems achieve flexibility by clearly separating the stable parts of the system (i. . the components) from the specification of their composition.

Components are black-box entities that encapsulate services behind well-defined interfaces. These interfaces tend to be very restricted in nature, reflecting a particular model of plug-compatibility supported by a component-framework[5]. The Figure 1 above compares the traditional development and CB development. The new model leverages the fact that real-time software can be implemented as a multitasking application, thus the driver itself can have its own thread of control.

The driver uses a data-driven approach in (Figure 1b) to interact with the rest of the application software, as opposed to the more traditional OO process driven approach shown in (Figure 1a). The real power of CB methods comes from the reusability of software component. The idea of reusing software embodies several advantages. It improves productivity, maintainability, portability and quality of software systems.

A reusable component can be seen as a box, which contains the code and the documentation .

Component can be Black Box, Glass-box or, white box in nature. Most of the COTS components are black box or glass-box in nature. In black box reuse, the reuse sees the interface, not the implementation of the component.

The interface contains public methods, user documentation, requirements and restrictions of the component. In glass box reuse, the inside of the box can be seen as well as the outside, but it is not possible to touch the inside. This solution has an advantage when compared to black box reuse, as the reuse can understand the box and, its use better.

The importance of component reusability depends on some quality factors such as existence of Meta-Information , rate of component’s observe-ability , rate of component’s customizability and, self- completeness of component’s return value[5].

The CB system development has become more difficult because of the increasing complexity, reduced development time and reduced budgets. So correctness and reliability are very important topics in this field of engineering [6]. State coverage and transition coverage are two popular State-based testing techniques.

Suman [4] has presented an approach to synthesize the state models of components without using its source code or other associated models. State models of the components are important for state-based regression testing of component-based applications. State-based bugs are difficult to detect using traditional testing techniques.

Bringmann [7] has proposed a test method using “ time partition” that enables the systematic definition of executable test cases for testing the continuous behaviour of component based automotive embedded systems.

This method is based on a graphical notation for test cases, which is easy to understand, and powerful enough to express complex, fully automated tests. Another Model based component regression testing method is proposed by Lalchandani [8] which covers dynamic slicing algorithm to discover the bugs in the architecture models produced over design time by converting them to Architecture Component Dependence Graph(ACDG). Several empirical studies were also conducted to highlight the component integration issues and techniques in component integration testing process[9].

Raheman[9] surveyed nearly all of the Component integration testing techniques such as 1) built-in testing (BIT); (2) testable architecture; (3) metadata-based; (4) certification strategy;(5) User’s specification-based testing.

Based on this finding, we can evolute any other component based integration testing technique, which are not invented yet. We can also compare difference /similarity between the methods. There may be various use of that categorization which depends on user of it. Selection of approach to test the component depends on various factors and assumptions.

But, until now, no previous work has been found in testing process area. Popular OO-metrics for system complexity calculation are code base such as lines of code, function point analysis, bugs or faults per lines of code, code coverage, number of lines of customer requirements, number of classes / interfaces. Using above metrics it is not possible to evaluate or measure the complexity of the system or component interaction complexities within the system until the coding process. And those metrics are inappropriate for calculating the functional importance of system’s module.

Those traditional OO-methodology based metrics focus on non-component based software systems and are inappropriate to CBSD mainly because of the nature of the COTS-components which are black box in nature due to size of the code or implementation details are not known in advance , whereas most of the traditional size metrics are based on line of codes. Lack of source code for some components prevents its comprehensive white box testing.

However, sometimes, it is possible to glass-box-test the CB-systems by using information such as design or functionality models.

Component metrics measures different aspects of CB-system Complexity; thereby extracting useful information about external quality aspects of system like maintainability, reusability and reliability [5, 47, 50]. Component metrics are used to evaluate properties of something being measured, such as quality, complexity or force-effort. Component complexity combines two measures: intrinsic complexities existed by methods inside the component and, extrinsic complexities existed by the resulting interactions with other components.